GPGPU via OpenCL et *boost::compute*

L'objectif de ce TP est d'utiliser la carte graphique comme unité de calcul fortement parallèle. L'idée est ici d'expérimenter autour d'OpenCL et de la programmation générique via *boost::compute*.

Exercise 1 : *Warm-up*

Dans un premier temps, récupérez le tuto du TP via git :

```
git clone https://github.com/dcoeurjo/BoostComputeTuto.git
```

Dans ce package, vous trouverez qq scripts de compilation `cmake` et un copie locale de *boost::compute*. Pour compiler un programme, soit vous passez par `cmake` dans un repertoire de build, soit vous utilisez la ligne de commande

```
g++ -o devices devices.cpp -Icompute/include -lOpenCL
```

ou

```
g++ -o devices devices.cpp /usr/lib/x86_64-linux-gnu/libOpenCL.so.1 -
Icompute/include
```

sur certaines machines du bâtiment.

1. Compilez et testez `devices`. Vous devriez avoir la liste des devices accessibles par OpenCL et *boost::compute*
2. Compilez et regardez l'exemple `apply.cpp`. Décrivez son fonctionnement.
3. Faites un petit tour dans la documentation de *boost::compute* (Documentation) et regardez quelques algorithmes et en particulier `exclusive\_scan`, `transform` ou encore `for\_each`.

4. Écrivez un petit programme en utilisant les algorithmes de *boost::compute* prenant en entrée un tableau $tab(i)$ de nombres (aléatoires) et retournant la valeur $val = \sum_i tab(i)^2$.

5. Écrivez un petit programme pour compter le nombre de valeurs impaires de $\{tab(i)^2\}$. Nous souhaitons récupérer dans un tableau annexe, l'ensemble des valeurs impaires de $\{tab(i)^2\}$.

6. (optionel) Plus difficile, comment calculer, sous forme de programmation générique : $val = \sum_{i=0}^{n} i \cdot tab(i)^2$? (regardez par exemple les itérateurs génériques de *boost::compute*, nottamenet les `zip_iterators`).

Exercise 2 : Intersection rayon-sphères

> **Remarques** L'approche générique est souvent à privilégier pour des petits problèmes de traitements de données. Cependant, *boost::compute* permet également de décrire des *microprogrammes* OpenCL s'appliquant sur des strucutres *boost::compute*. Par exemple, il est tout à fait possible d'avoir des types `compute::vector<Sphere>` sur des structures (`Sphere`) complexes. C'est parfois un peu pénible et on pourrait préferer des buffers indépendants (rayons, position du centre...) que l'on passerait à un *kernel* OpenCL. Regardez `applyKernel.cpp` pour un exemple de cela. Nous vous conseillons cette approche pour certaines étapes de cet exercise.

On cherche maintenant à résoudre le problème suivant : étant donné un ensemble de N sphères 3D et nous souhaitons connaître la liste des sphères en intersection avec un certain rayon, et ce sur GPU.

Chaque sphère est donnée par un centre dans l'espace et un rayon. Dans un premier temps, vous pouvez implémentez le sous-problème suivant: étant donné un ensemble de sphères, trouver le sous-ensemble contenant un certain point p. Cela ne change en rien l'algorithme, ça évite l'étape du calcul d'intersection rayon-sphère (pas très compliqué par ailleurs). Dans la mesure du possible et si cela n'est pas précisé, toutes les données doivent rester sur le GPU. On souhaite également que chaque traitement se fasse en un minimum d'instructions génériques. Pour vérifier l'exactitude de vos calculs, vous pouvez, en parallèle, faire le calcul également sur CPU (générique ou non) et comparer les résultats.

1. Écrivez un petit generateur de sphère aléatoires, sur le GPU, avec la fonction `generate`. On appelle $\mathcal{S} = \{S_i\}$ la liste des sphères ainsi générées (cf remarque précédente, vous pouvez avoir des buffers indépendants pour le rayon et les positions x, y, z).

2. Écrivez le programme permettant de construire un tableau de booléan `isinside` tel que `isinside\[i\]` est vrai si le point p est dans la sphère S_i.

3. Modifiez votre programme pour qu'il retourne le nombre de sphères contenant le point p.
4. Nous souhaitons maintenant construire un tableau **ne contenant que les sphères contenant p.**
5. Modifiez votre programme pour cette fois avoir une intersection rayon-sphères. Mesurez les performances de votre systèmes.
6. Nous souhaitons maintenant obtenir, sur le CPU, la liste des sphères, intersectées par le rayon, ordonnées selon la distance à l'origine du rayon.